

### In the Specification

Please replace the paragraph beginning on page 7, line 25 and ending on page 8, line 3 with the following amended paragraph.

Allowing the ~~reuse~~ reuse of subcircuitry for different instruction types provides other benefits as well. By reusing subcircuitry, support for changes in the instruction set is provided. New instructions can be added and flexibly handled using the techniques and mechanisms of the present invention. For example, when smaller instructions such as 16-bit instructions are introduced into a 32-bit instruction set, the alignment restrictions are typically reduced to match the size of the smallest instruction. Using the techniques and mechanisms of the present invention, the branch instruction will be able to jump to these instructions even if the new instructions are not multi-byte aligned. For example, if instructions in instruction set were previously word aligned, the new instructions can be byte aligned or half word aligned and the branch instruction will still be able to jump to the appropriate memory location.

Please replace the paragraph beginning on page 8, line 19 and ending on page 8, line 32 with the following amended paragraph.

Figure 1 is a diagrammatic representation showing a processor configured to handle an instruction set. A processor 101 includes a register bank with register 111, register 113, and register 115. Processor 101 also includes circuitry 131 for handling various instructions. Circuitry 131 includes subcircuitry 141. Any portion of processor circuitry used to perform instructions in an instruction set is referred to herein as subcircuitry. Processor 101 is connected to memory 151 with memory lines 153-161. It should be noted that a processor 101 may also have other configurations and include other components not shown such as ~~[[and]]~~ an instruction cache, data cache, and ~~[[were]]~~ a memory controller. In typical examples, a processor reads instructions sequentially from memory and performs these instructions. The instructions may involve operations on data also loaded from memory and placed in registers for even more efficient access. Performing operations such as arithmetic operations involves using particular hardware within the processor 101. A branch instruction may involve using another piece of hardware within processor 101.

Please replace the paragraph beginning on page 9, line 8 and ending on page 9, line 23 with the following amended paragraph.

According to various embodiments of the present invention, branch instructions allow the processing of a sequence of operations to move to a different multi-byte aligned memory location. In one example, instructions are word aligned. The branch instruction allows a branch to a particular word aligned address. However, branches to other addresses are not allowed. In general, it will be appreciated that restricting alignment to multi-byte or word aligned memory locations allows access to a greater range of memory addresses. More specifically, a limited number of bits are available to store the offset associated with a branch. In one simplified example, three bits or eight different values of offset are provided. If no multi-byte alignment is used, the branch target address is restricted to a three bit range of eight bytes in the memory space. However, if word alignment is used, the offset is multiplied by 4. The branch target address range is now expanded to a five bit range or thirty-two bytes in the memory space. In this example, the target address still can only be in  $2^5$  memory locations, however the eight specific locations are spread over a range of thirty-two bytes instead of being limited to a range of eight bytes.

Please replace the paragraph beginning on page 11, line 21 and ending on page 11, line 23 with the following amended paragraph.

Techniques and mechanisms of the present invention allow the units of branch instructions to be in bytes even though many of the instructions are multiple bytes in size and  $[a]$  required to be aligned on multi-byte addresses.

Please replace the paragraph beginning on page 16, line 1 and ending on page 16, line 15 with the following amended paragraph.

Figure 8 is a diagrammatic representation showing one example of a system using secondary side arbitration, sometimes referred to as slave side arbitration, simultaneous multiple

primary components, or simultaneous multiple masters. A system using individual arbitrators that correspond to individual secondary components accessible by more than one primary component is referred to herein as a secondary side arbitration system. The secondary side arbitration system no longer requires a bus or a system bus arbitrator that prevents a second primary component from accessing a second secondary component when a first primary component is accessing a first secondary component. According to various embodiments, a secondary component such as ~~peripheral interface~~ a memory 825 is associated with a secondary side arbitrator 851. However, secondary components UART 821 and PIO 823 are not associated with any arbitrator. In one example, secondary component UART 821 and secondary PIO 823 can only be accessed by primary CPU 813 and not by primary Ethernet device 815. A secondary memory component 825, however, can be accessed by both primary CPU 813 and primary Ethernet device 815.

Please replace the paragraph beginning on page 17, line 18 and ending on page 18, line 2 with the following amended paragraph.

In typical implementations, the generator program 905 can identify the selections and generate a logic description with information for implementing the various modules. The generator program 905 can be a Perl script creating HDL files such as Verilog, Abel, VHDL, and AHDL files from the module information entered by a user. In one example, the generator program identifies a portion of a high-level language program to accelerate. The other code is left for execution on a processor core. According to various embodiments, the generator program 905 identifies pointers and provides ports for each pointer. One tool with generator program capabilities is System on a Programmable Chip (SOPC) Builder available from Altera Corporation of San Jose, CA. The generator program 905 also provides information to a synthesis tool 907 to allow HDL files to be automatically synthesized. In some examples, a logic description is provided directly by a designer. Hookups between various components selected by a user are also interconnected by a generator program. Some of the available synthesis tools are Leonardo Spectrum, available from Mentor Graphics Corporation of Wilsonville, Oregon and Synplify available from Synplicity Corporation of Sunnyvale, California. The HDL files may

contain technology specific code readable only by a synthesis tool. The HDL files at this point may also be passed to a simulation tool [[909]].